University of California Santa Cruz
Department of Computer Engineering
Lab Experiment Report # 3
Two's Complement and LED Display

Author: Kyle Jeffrey
Lab Partner: NA
Due Date: 1/1/18

Objective

The purpose of this lab was to use combinational logic using mux structures and to familiarize with buses and bit vectors. The board will display two hex digits loaded by the switches and its two's complement if a button is pressed.

Kyle Jeffrey
Prof Schlag
CE 100
1/18/18

Module: Multiplexores

Most of the modules in this lab will be using these 3 mux modules:

- **4 to 1 mux**

```
e & (~sel[1] & ~sel[0] & in[0]) |
    (~sel[1] &  sel[0] & in[1]) |
    ( sel[1] & ~sel[0] & in[2]) |
    ( sel[1] &  sel[0] & in[3]) ;
```

- **8 to 1 mux**

```
e & ((~sel[2] & ~sel[1] & ~sel[0] & in[0]) |
     (~sel[2] & ~sel[1] &  sel[0] & in[1]) |
     (~sel[2] &  sel[1] & ~sel[0] & in[2]) |
     (~sel[2] &  sel[1] &  sel[0] & in[3]) |
     ( sel[2] & ~sel[1] & ~sel[0] & in[4]) |
     ( sel[2] & ~sel[1] &  sel[0] & in[5]) |
     ( sel[2] &  sel[1] & ~sel[0] & in[6]) |
     ( sel[2] &  sel[1] &  sel[0] & in[7]) );
```

- **2 to 1x8 mux(two 8 bit inputs, one 8 bit output)**

```
o[7] =  in0[7] & ~sel | in1[7] & sel;
o[6] =  in0[6] & ~sel | in1[6] & sel;
o[5] =  in0[5] & ~sel | in1[5] & sel;
o[4] =  in0[4] & ~sel | in1[4] & sel;
o[3] =  in0[3] & ~sel | in1[3] & sel;
o[2] =  in0[2] & ~sel | in1[2] & sel;
o[1] =  in0[1] & ~sel | in1[1] & sel;
o[0] =  in0[0] & ~sel | in1[0] & sel;
```

Method**:**

The mux has inputs n and requires selector m inputs such that $2^m = n$. In short, there should be enough selector permutations to pick each input. All of these have been done with simple Sum of Products.

Kyle Jeffrey
Prof Schlag
CE 100
1/18/18

Module: 7-Segment Display

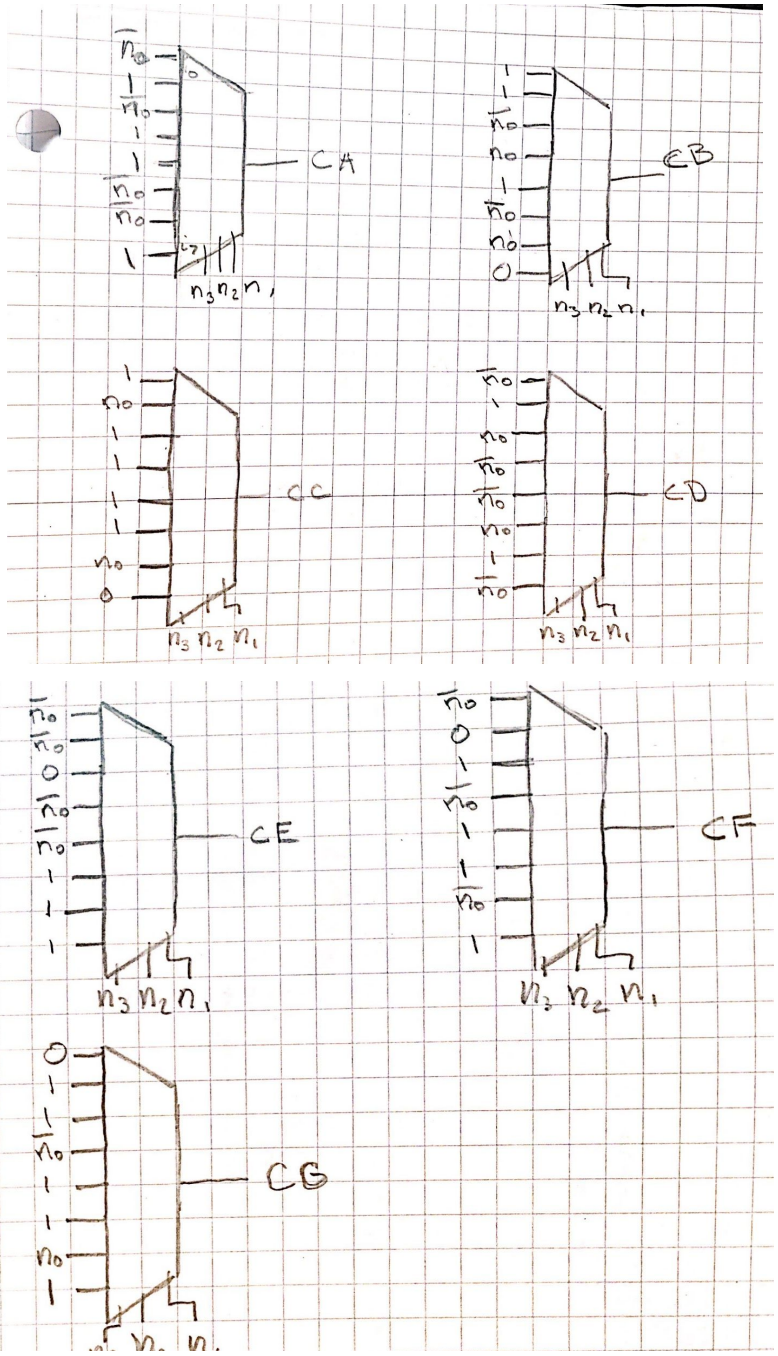
The 7 segment display has inputs CA - CG, 7 inputs, each powering a different segment on the LED display, requiring that I take a 4 bit vector (the sum of our switch inputs) and converting it to a 7-bit vector corresponding to CA through CG. The inputs an[3:0] control which of the 4 hex displays our segments are powering, and in this lab we only cared about an0 and an1.

n3	n2	n1	n0	CA	CB	CC	CD	CE	CF	CG
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0	1	1	1	0	1	1	1
1	0	1	1	0	0	1	1	1	1	1
1	1	0	0	1	0	0	1	1	1	0
1	1	0	1	0	1	1	1	1	0	1
1	1	1	0	1	0	0	1	1	1	1
1	1	1	1	1	0	0	0	1	1	1

Method**:**

The four bit number generates sixteen possible permutations that correspond to a number to be displayed on the LED. I began by writing out a truth table with the 4 inputs and 7 outputs. 7 outputs, CA through CG, are tied to 8 to 1 muxes. The selector bits on the mux are tied to n3,

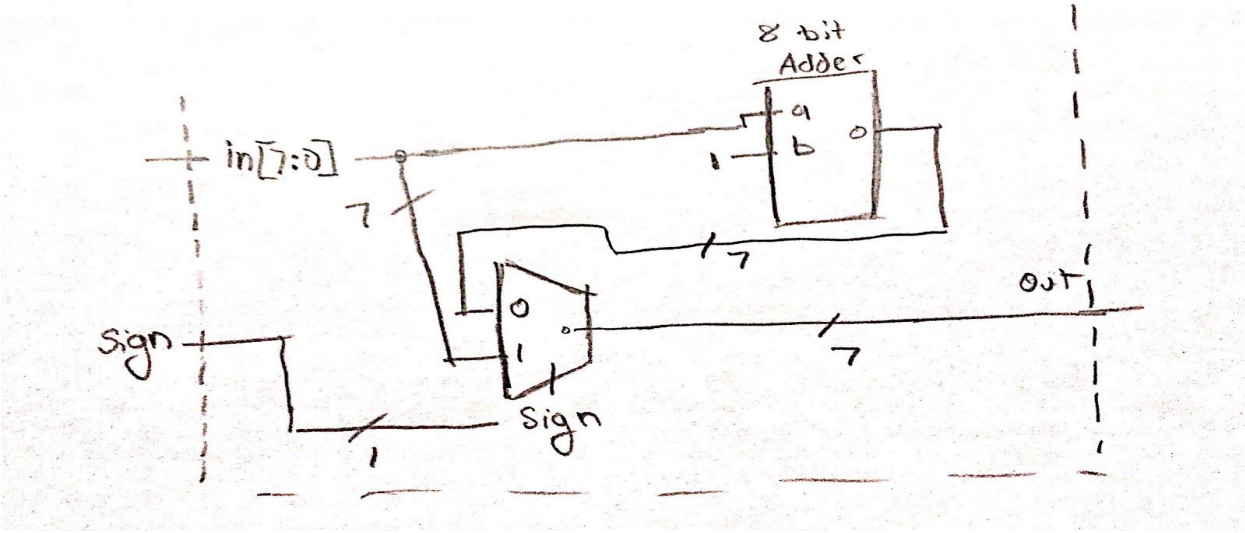
n2, and n1. The possible inputs for the mux are 0, 1, n0 or ~n0.To determine each input, determine how n0 relates to the segment using these possible inputs.



Kyle Jeffrey
Prof Schlag
CE 100
1/18/18
5

Module: Sign Changer

The sign changer has two inputs: the 8 bit number to be changed, and a 1 bit input deciding whether to output the two's complement of the input or the input itself. The module is comprised of an 8 bit adder made of muxes and a mux tied to the inputs determining which number to output. The adder in the design was hard coded to add 1 to an inverted input.



.

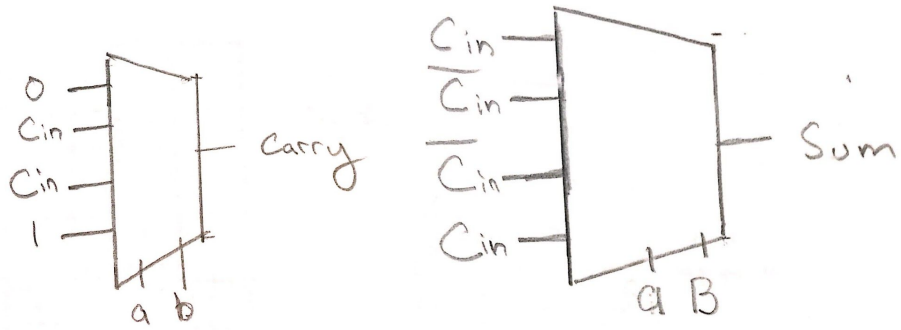
Method**:**

Full Adder: Each full adder is made with 2, 4-1 muxes. This meant that were 2 selector pins like the logic for the hex display. Using the same strategy as I did with the hex display I created truth table and determined what the sum and carryout would be, hence the two muxes (1 for each output).

n1	n0	cin	sum	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1

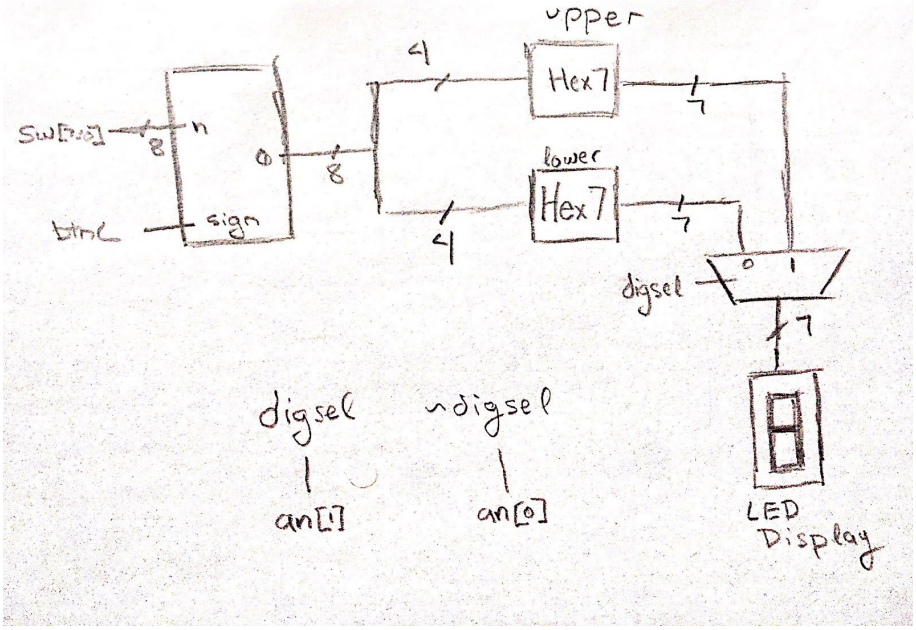
Kyle Jeffrey
Prof Schlag
CE 100
1/18/18
6

1	1	0	0	1
1	1	1	1	1



8-bit Adder: This adder is comprised of 8 rippled full adder's. When rippling the adder's the first bit adder should have its carry in set to 0 to simulate a half adder. Each subsequent adder had its carry out tied to the next carry in, and the final adder had a carry out set to a disconnected wire Part of getting the adder to do two's complement is having an overflow bit is not display.

Top Level Design:



Kyle Jeffrey
Prof Schlag
CE 100
1/18/18

The top level module had inputs sw[7:0], btnC, btnR, and clkin, and outputs seg[6:0], dp, an[3:0]. The switches are directly attached to the sign changer and represent the input value in binary. There are two instances of the hex display to display the 8 bit output. A provided module included an input called digsel that controlled the frequencies at which the displays flickered. There is only actually 7 inputs for all 4 LED displays, and to actually display on all 4, the module needs to swap what value is being displayed and which LED is activated.

Test Bench and Simulation:

Once downloading the given testbench file and correctly changing all of the i/o names for simulation, I began simulation debugging. Using the provided code, the clock was simulated.

Method:

I used the simulation to make sure that:

1. The adder was adding correctly
2. The Sign Changer output the correct selection
3. The display logic printed the correct hex digit on the right display

Testing the adder: The adder was tested by setting the switches to at least eight different values. Once testing eight of the values, I had confidence that the logic was correct. The most important values were fifteen and sixteen which acted as boundary cases. If those both worked, then the middle values were also more likely to work. In practice, this was tested by setting sw[6:0] to different values and seeing what the wires s0, s1, and s2 equaled.

Testing the Hex Display: The hex display was less intuitive to debug. This was debugged at the same time as debugging the adder, but the logic for this was much easier to have a mistake on because of the number of equations. So, using the switches and going through values zero to sixteen was necessary in determining if the display was correct.

RESULTS:

Question 1: How fast the signal **dig_sel** is oscillating.

Answer: The period of dig_sel was roughly 320 ns, meaning it would switch from low to high and vice versa every 160 ns.

Kyle Jeffrey
Prof Schlag
CE 100
1/18/18
8

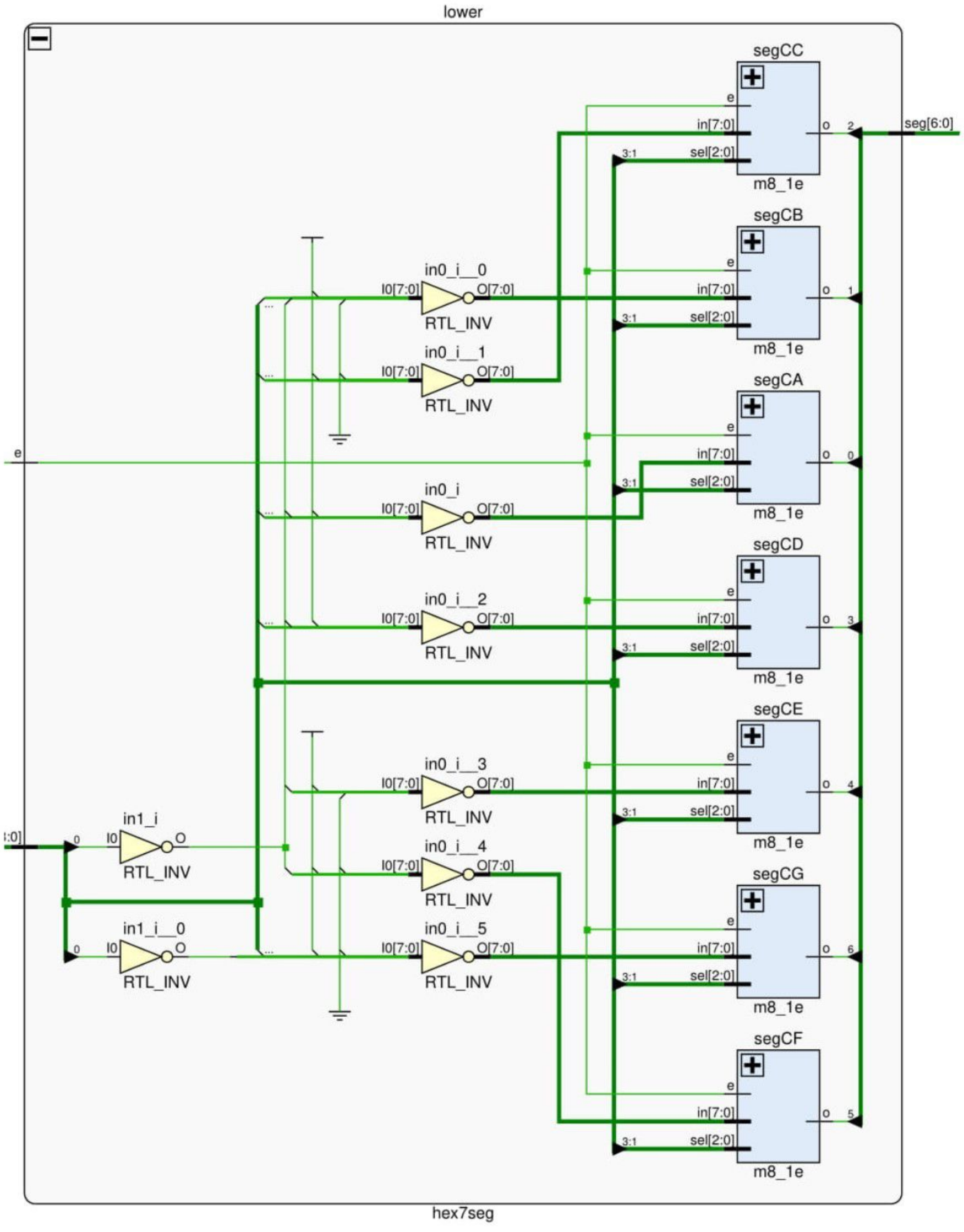
Question 2: Whether you observed any flickering in the 7-segment display.

Answer: I wasn't actually able to observe any flickering in the LED's.

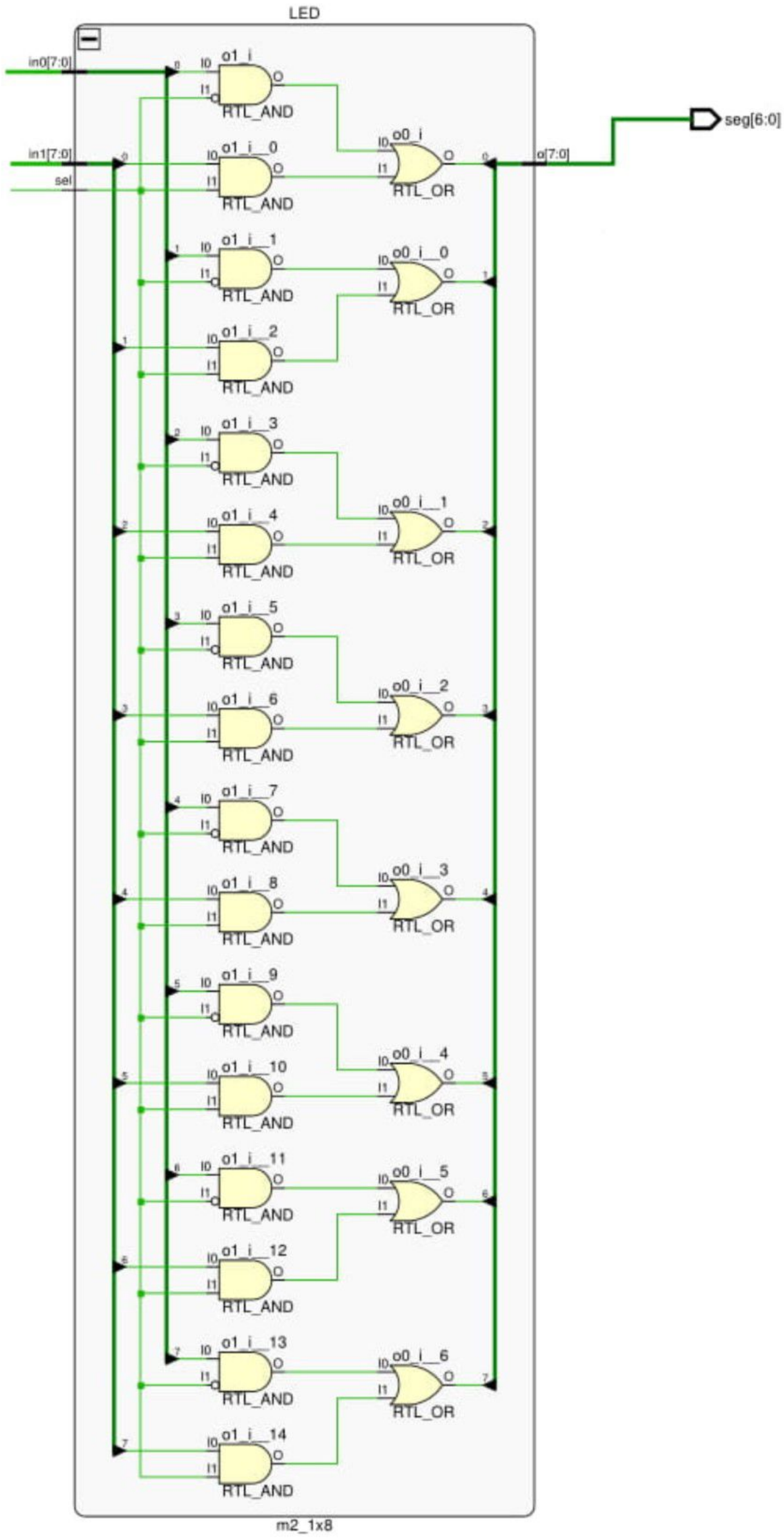
Conclusion

This lab was an introduction into mux logic design. It also ventured into new perspectives on truth tables and dabbled lightly into K-maps. There weren't too many errors when the project was implemented on the board, but messy notes did lead to issues with displaying the hex characters. Cleaner notes would be paramount next time. The sign changer probably could've been optimised to use the sign input as a 1 to input to the adder or a 0.

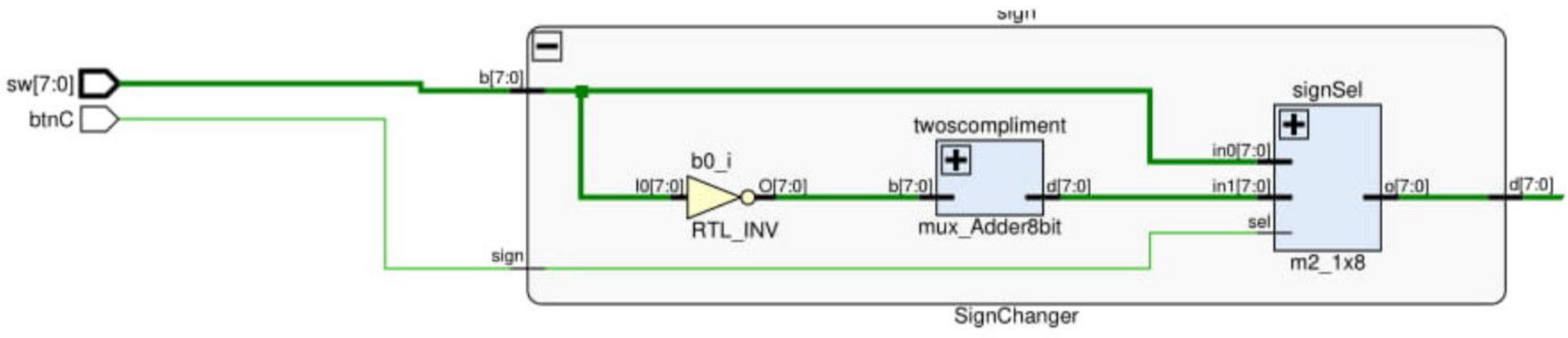
Hex Logic


lower
segCC
e
in[7:0]
sel[2:0]
3:1
o
2
seg[6:0]
m8_1e
segCB
1
segCA
0
segCD
3
segCE
4
segCG
6
segCF
5
in0_i__0
in0_i__1
in0_i
in0_i__2
in0_i__3
in0_i__4
in0_i__5
in1_i
in1_i__0
I0[7:0]
O[7:0]
I0
O
RTL_INV
hex7seg

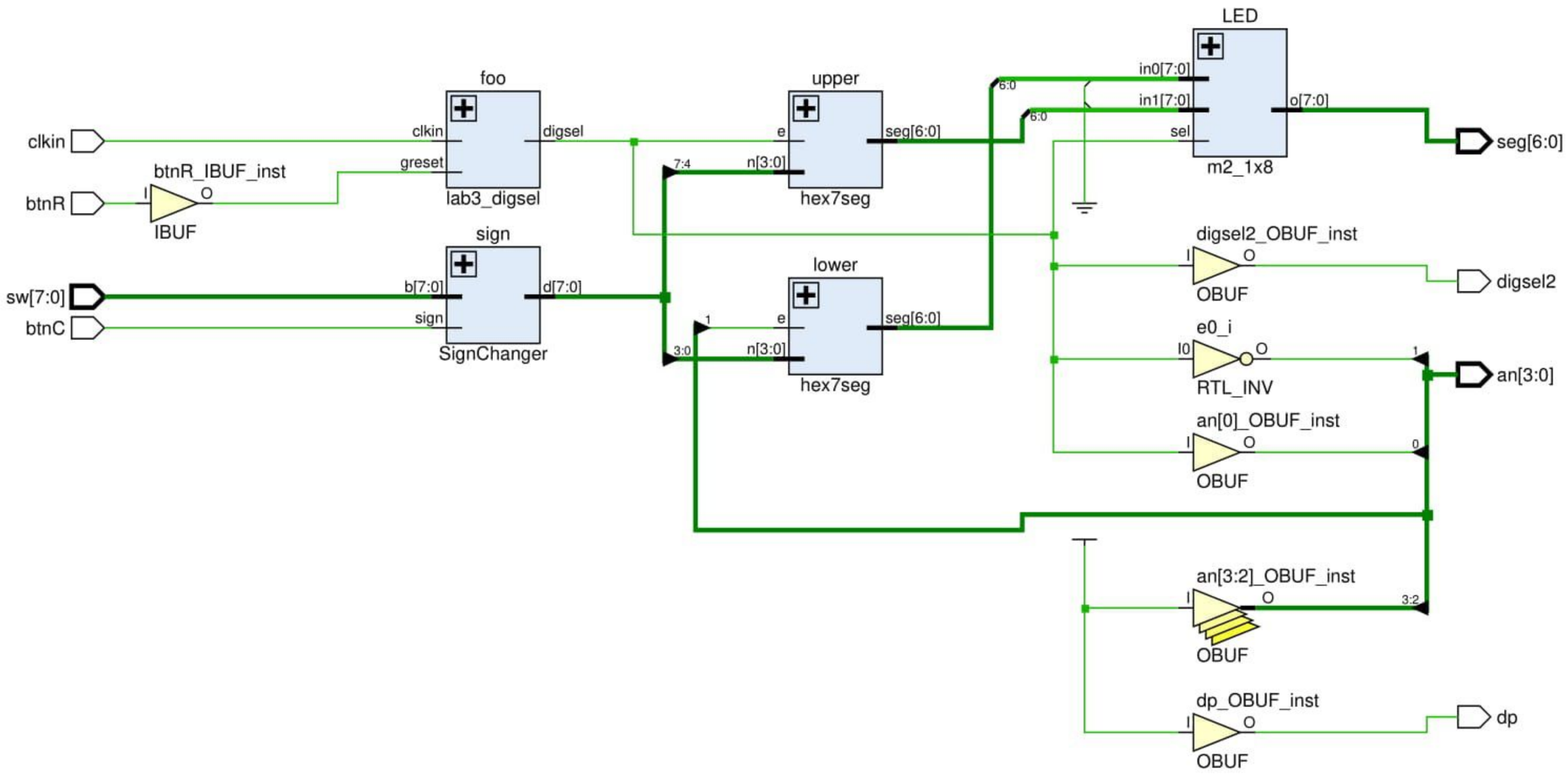
Mux Sign Selector



Sign Changer



Top Module


LED
in0[7:0]
in1[7:0]
sel
o[7:0]
m2_1x8
seg[6:0]
foo
clkin
greset
digsel
lab3_digsel
upper
e
n[3:0]
seg[6:0]
hex7seg
clkin
btnR_IBUF_inst
I
O
IBUF
btnR
sign
b[7:0]
sign
d[7:0]
SignChanger
sw[7:0]
btnC
lower
e
n[3:0]
seg[6:0]
hex7seg
7:4
1
3:0
6:0
6:0
digsel2_OBUF_inst
OBUF
digsel2
e0_i
I0
O
RTL_INV
1
an[3:0]
an[0]_OBUF_inst
OBUF
0
an[3:2]_OBUF_inst
OBUF
3:2
dp_OBUF_inst
OBUF
dp

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01/21/2019 02:10:37 PM
// Design Name:
// Module Name: mux_Adder8bit
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module mux_Adder8bit(
    input [7:0] b,
    output [7:0] d
    );
    wire [7:0] p;
        // to change the sign, flip the input of b and add 1, OVERFLOW IN WIRE P7
    mux_Adder s0(.a(b[0]), .b(1'b1), .cin(1'b0), .cout(p[0]), .s(d[0]) );
    mux_Adder s1(.a(b[1]), .b(1'b0), .cin(p[0]), .cout(p[1]), .s(d[1]) );
    mux_Adder s2(.a(b[2]), .b(1'b0), .cin(p[1]), .cout(p[2]), .s(d[2]) );
    mux_Adder s3(.a(b[3]), .b(1'b0), .cin(p[2]), .cout(p[3]), .s(d[3]) );
    mux_Adder s4(.a(b[4]), .b(1'b0), .cin(p[3]), .cout(p[4]), .s(d[4]) );
    mux_Adder s5(.a(b[5]), .b(1'b0), .cin(p[4]), .cout(p[5]), .s(d[5]) );
    mux_Adder s6(.a(b[6]), .b(1'b0), .cin(p[5]), .cout(p[6]), .s(d[6]) );
    mux_Adder s7(.a(b[7]), .b(1'b0), .cin(p[6]), .cout(p[7]), .s(d[7]) );

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01/21/2019 01:42:42 PM
// Design Name:
// Module Name: mux_Adder
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module mux_Adder(
    input a,b,cin,
    output s, cout
    );

    m4_1e sum(.in({cin,~cin,~cin,cin}), .sel({a,b}), .e(1'b1), .o(s));

    m4_1e carry(.in({1'b1, cin, cin, 1'b0}), .sel({a,b}), .e(1'b1), .o(cout));
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01/21/2019 02:43:56 PM
// Design Name:
// Module Name: hex7seg
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module hex7seg(
    input [3:0] n,
    input e,
    output [6:0] seg
    );  // not the mux input because the LED is active low
    //                                          input     7         6         5       4
     3     2         1       0
    m8_1e segCA(.e(e), .sel(n[3:1]), .o(seg[0]), .in(~{ 1'b1,   ~n[0],   ~n[0],
1'b1,   1'b1,  n[0],   1'b1,  ~n[0]}));
    m8_1e segCB(.e(e), .sel(n[3:1]), .o(seg[1]), .in(~{ 1'b0,    n[0],   ~n[0],
1'b1,   n[0], ~n[0],  1'b1,   1'b1}));
    m8_1e segCC(.e(e), .sel(n[3:1]), .o(seg[2]), .in(~{ 1'b0,    n[0],    1'b1,
1'b1,   1'b1,  1'b1,  n[0],   1'b1}));
    m8_1e segCD(.e(e), .sel(n[3:1]), .o(seg[3]), .in(~{~n[0],    1'b1,    n[0],
~n[0],  ~n[0],  n[0],  1'b1,  ~n[0]}));
    m8_1e segCE(.e(e), .sel(n[3:1]), .o(seg[4]), .in(~{ 1'b1,    1'b1,    1'b1,
~n[0],  ~n[0],  1'b0, ~n[0],  ~n[0]}));
    m8_1e segCF(.e(e), .sel(n[3:1]), .o(seg[5]), .in(~{ 1'b1,   ~n[0],    1'b1,
1'b1,  ~n[0],  1'b1,  1'b0,  ~n[0]}));
    m8_1e segCG(.e(e), .sel(n[3:1]), .o(seg[6]), .in(~{ 1'b1,    n[0],    1'b1,
1'b1,  ~n[0],  1'b1,  1'b1,   1'b0}));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01/20/2019 05:36:26 PM
// Design Name:
// Module Name: m2_1x8
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m2_1x8(
    input [7:0] in0,
    input [7:0] in1,
    input sel,
    output [7:0] o
    );
     assign o[7] =  in0[7] & ~sel | in1[7] & sel;
     assign o[6] =  in0[6] & ~sel | in1[6] & sel;

     assign o[5] =  in0[5] & ~sel | in1[5] & sel;
     assign o[4] =  in0[4] & ~sel | in1[4] & sel;
     assign o[3] =  in0[3] & ~sel | in1[3] & sel;
     assign o[2] =  in0[2] & ~sel | in1[2] & sel;
     assign o[1] =  in0[1] & ~sel | in1[1] & sel;
     assign o[0] =  in0[0] & ~sel | in1[0] & sel;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01/20/2019 05:36:26 PM
// Design Name:
// Module Name: m2_1x8
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m4_1e(
    input [3:0] in,
    input [1:0] sel,
    input e,
    output o
    );

    assign o = e & (~sel[1] & ~sel[0] & in[0]) |
                   (~sel[1] &  sel[0] & in[1]) |
                   ( sel[1] & ~sel[0] & in[2]) |
                   ( sel[1] &  sel[0] & in[3]) ;

endmodule
```

```
module m8_1e(
    input [7:0] in,
    input [2:0] sel,
    input e,
    output o
    );
     assign o = e & ((~sel[2] & ~sel[1] & ~sel[0] & in[0]) |
                     (~sel[2] & ~sel[1] &  sel[0] & in[1]) |
                     (~sel[2] &  sel[1] & ~sel[0] & in[2]) |
                     (~sel[2] &  sel[1] &  sel[0] & in[3]) |
                     ( sel[2] & ~sel[1] & ~sel[0] & in[4]) |
                     ( sel[2] & ~sel[1] &  sel[0] & in[5]) |
                     ( sel[2] &  sel[1] & ~sel[0] & in[6]) |
                     ( sel[2] &  sel[1] &  sel[0] & in[7]) );


endmodule
```

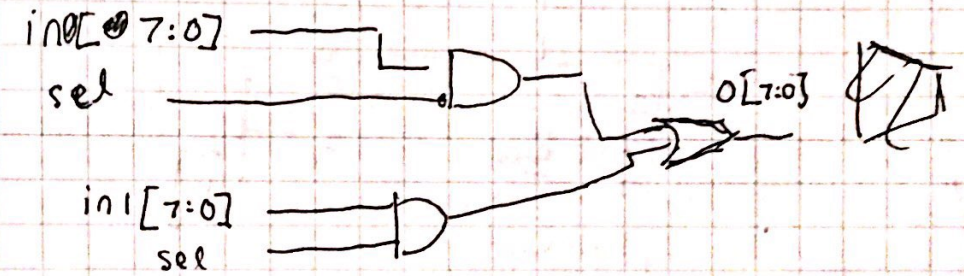
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01/20/2019 05:36:26 PM
// Design Name:
// Module Name: m2_1x8
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m2_1x8(
    input [7:0] in0,
    input [7:0] in1,
    input sel,
    output [7:0] o
    );
     assign o[7] =  in0[7] & ~sel | in1[7] & sel;
     assign o[6] =  in0[6] & ~sel | in1[6] & sel;

     assign o[5] =  in0[5] & ~sel | in1[5] & sel;
     assign o[4] =  in0[4] & ~sel | in1[4] & sel;
     assign o[3] =  in0[3] & ~sel | in1[3] & sel;
     assign o[2] =  in0[2] & ~sel | in1[2] & sel;
     assign o[1] =  in0[1] & ~sel | in1[1] & sel;
     assign o[0] =  in0[0] & ~sel | in1[0] & sel;

endmodule
```

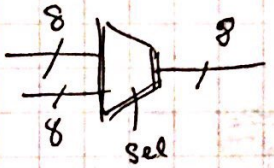
LAB 3

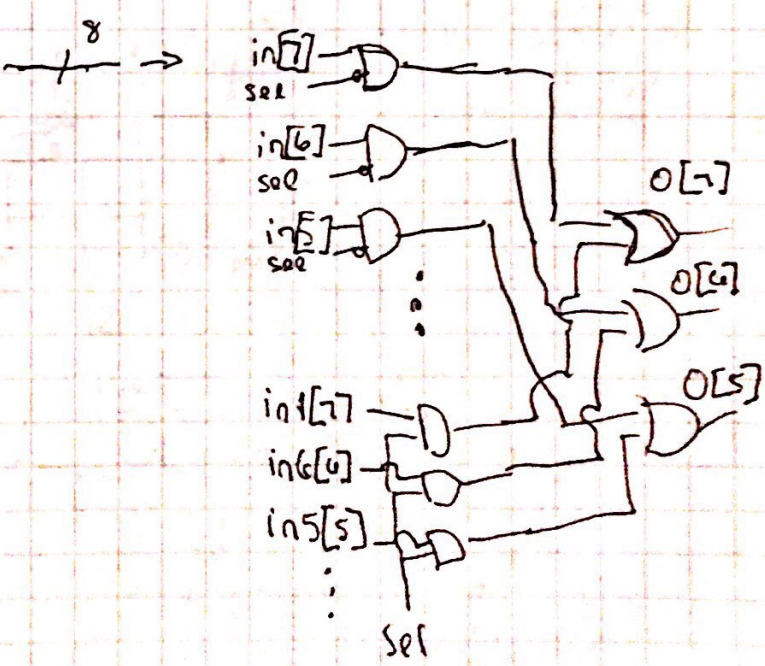
mux 4:1

$f(S_0, S_1, in0, in1, in2, in3) = \bar{S_0}\bar{S_1}in0 + \bar{S_1}S_0 in1 + S_1\bar{S_0} in2 + S_1 S_0 in3$

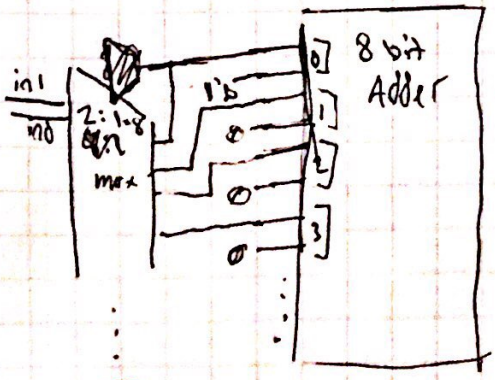
Mux 2:1 x 8 → A 2 to 1 mux, but the inputs are 8 bit







Sign Changer



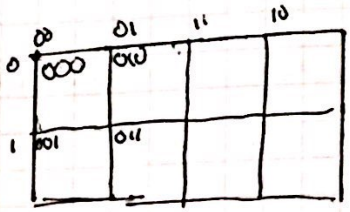
Sign Changer

→ implement 8 bit Adder
- Two 4 to 1 muxs
- inverter
- 0 & 1 as needed

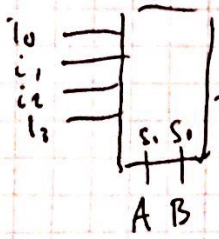
□ 1 bit Adder w/ 4:1 muxs

A B C_{in}	S C_o
0 0 0	0 0
0 0 1	1 0
0 1 0	1 0
0 1 1	0 1
1 0 0	1 0
1 0 1	0 1
1 1 0	0 1
1 1 1	1 1

Sum K-map

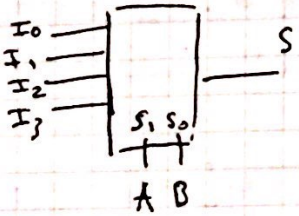


Carry Mux



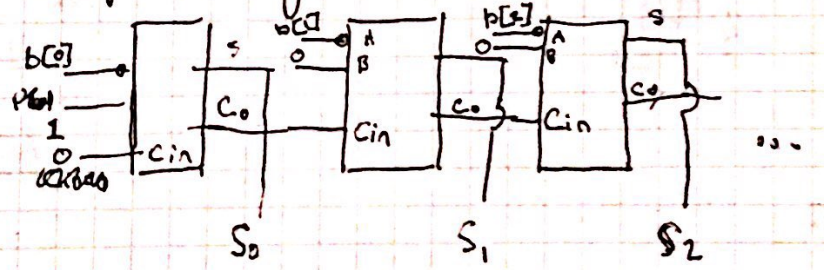
$i_0 = 0$
$i_1 = C_{in}$
$i_2 = C_{in}$
$i_3 = 1$

Sum mux



$I_0 = C_{in}$
$I_1 = \overline{C_{in}}$
$I_2 = \overline{C_{in}}$
$I_3 = C_{in}$

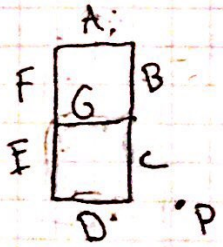
→ Sign Change Adder



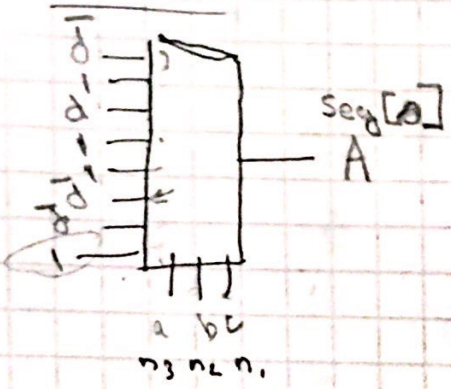
7 Segment Display

seg[0]

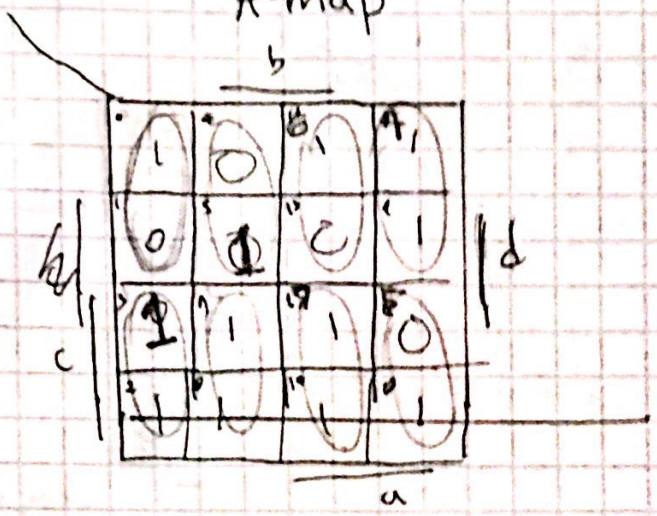
n_3 n_2 n_1 n_0	A B C D E F G
0 0 0 0	1 1 1 1 1 1 0
0 0 0 1	0 1 1 0 0 0 0
0 0 1 0	1 1 0 1 1 0 1
0 0 1 1	1 1 1 1 0 0 1
0 1 0 0	0 1 1 0 0 1 1
0 1 0 1	1 0 1 1 0 1 1
0 1 1 0	1 0 1 1 1 1 1
0 1 1 1	1 1 1 0 0 0 0
1 0 0 0	1 1 1 1 1 1 1
1 0 0 1	1 1 1 0 0 1 1
1 0 1 0	1 1 1 0 1 1 1
1 0 1 1	0 0 1 1 1 1 1
1 1 0 0	1 0 0 1 1 1 0

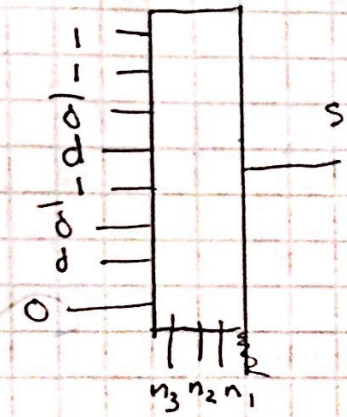


8×1 Mux

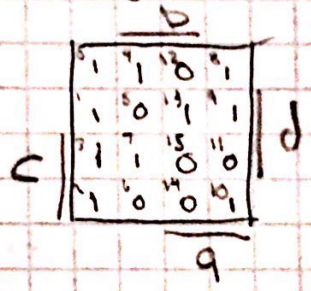


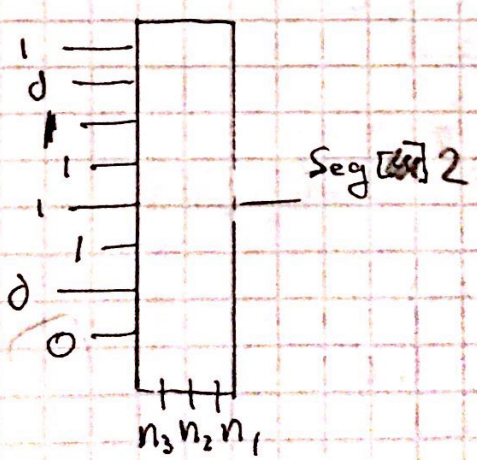
A-map



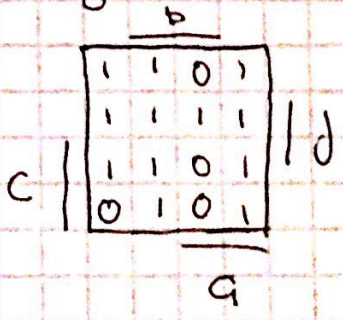


Seg[1] - Map (B)



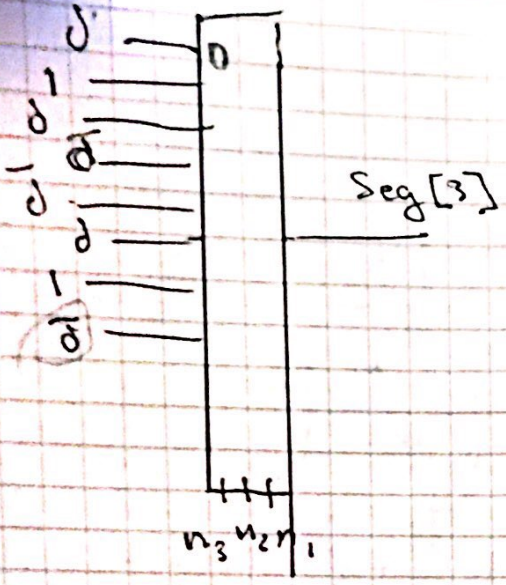


Seg[2] K-Map (C)

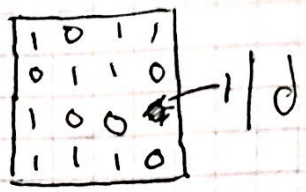


$d = n_0$

Bilalod

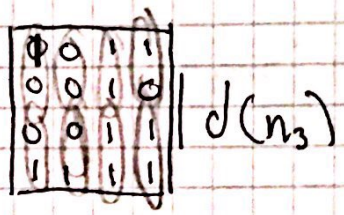


Seg[3] K-Map (D)



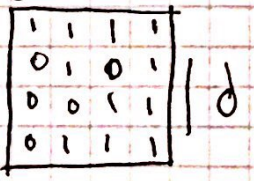
4

Seg[4] k-map (E)



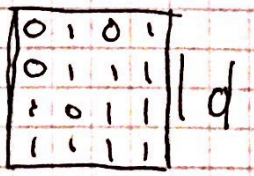
5

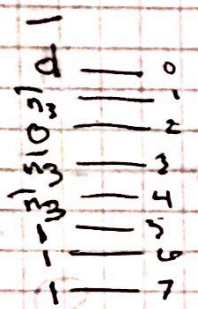
Seg[5] k-map (F)

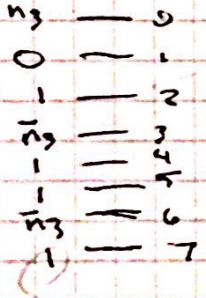


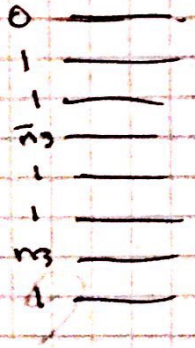
6

Seg[6] k-map (G)

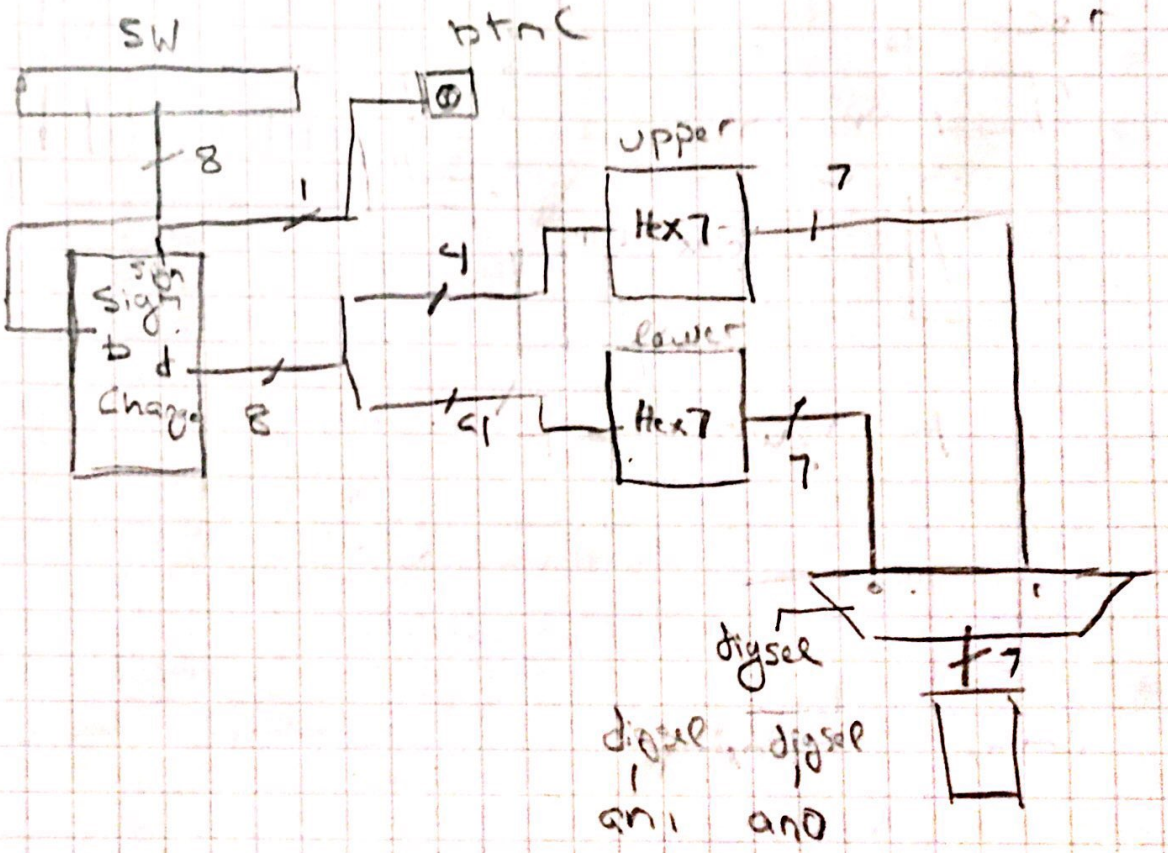








Top Level Module



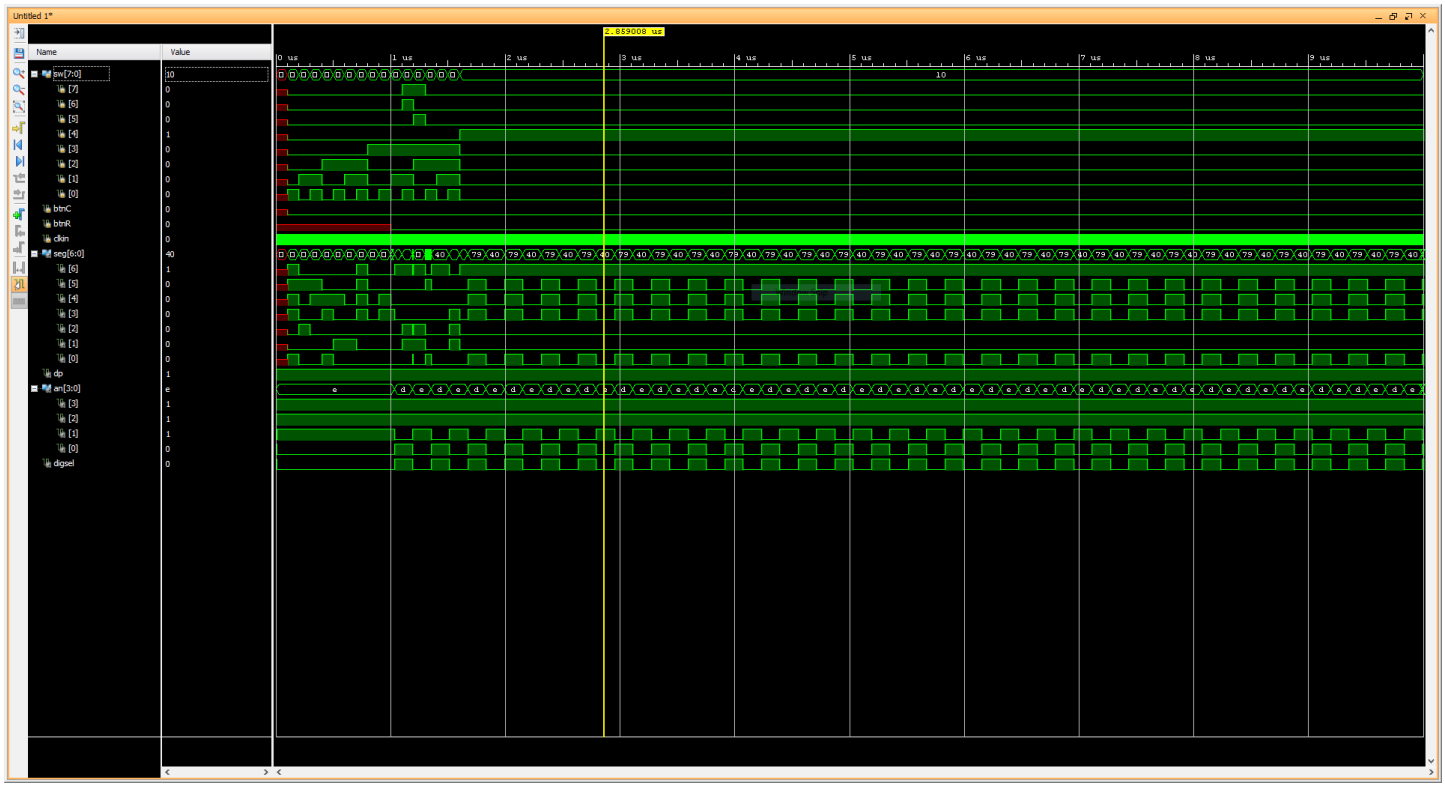
Lab 3: 7721

~~1-19~~ 1-24-19 10:42 AM

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01/20/2019 06:10:03 PM
// Design Name:
// Module Name: SignChanger
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module SignChanger(
    input sign,
    input [7:0] b,
    output [7:0] d
    );
    wire [7:0] compliment;


    mux_Adder8bit twoscompliment(.b(~b), .d(compliment));
    m2_1x8 signSel(.sel(sign), .in1(compliment), .in0(b), .o(d));


    //mux for sending input through to adder or going straight to output

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01/23/2019 02:54:38 PM
// Design Name:
// Module Name: topModule
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module topModule(
    input [7:0] sw,
    input btnC, btnR, clkin,
    output [6:0] seg,
    output dp,
    output [3:0] an,
    output digsel2
    );
    wire [7:0] binaryAnswer;
    wire [6:0] ledCorrectedUpper, ledCorrectedLower;
    wire digsel;

    assign digsel2 = digsel;
    SignChanger sign(.b(sw), .d(binaryAnswer), .sign(btnC));
    hex7seg lower(.n(binaryAnswer[3:0]), .seg(ledCorrectedLower), .e(~digsel));
    hex7seg upper(.n(binaryAnswer[7:4]), .seg(ledCorrectedUpper), .e(digsel));
    m2_1x8 LED(.in0({1'b0, ledCorrectedLower}), .in1({1'b0,ledCorrectedUpper}),
.sel(digsel), .o(seg));

    assign an[3] = 1'b1;
    assign an[2] = 1'b1;
    assign dp = 1'b1;

    assign an[1:0] = {~digsel, digsel};
```

```
    lab3_digsel foo(.digsel(digsel), .clkin(clkin), .greset(btnR));


endmodule
```


Untitled 1*
Name
Value
sw[7:0]
10
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
btnC
btnR
clkin
seg[6:0]
40
[6]
[5]
[4]
[3]
[2]
[1]
[0]
dp
an[3:0]
e
[3]
[2]
[1]
[0]
digsel
2.859008 us
0 us
1 us
2 us
3 us
4 us
5 us
6 us
7 us
8 us
9 us